

**EUROPEAN PATENT APPLICATION**

Application number: 86107713.9

Int. Cl.<sup>4</sup>: G 06 F 12/08

Date of filing: 06.06.86

Priority: 28.06.85 US 750381

Date of publication of application:  
04.02.87 Bulletin 87/6

Designated Contracting States:  
CH DE FR GB IT LI NL SE

Applicant: Hewlett-Packard Company  
Mail Stop 20 B-D 3000 Hanover Street  
Palo Alto California 94304(US)

Inventor: Worley, William S.  
19316 Falmouth Court  
Saratoga CA. 95070(US)

Inventor: Bryg, William R.  
18630 Perogo Way  
Saratoga CA. 95070(US)

Inventor: Baum, Allen  
2310 Cornell Street  
Palo Alto CA. 94306(US)

Representative: Liesegang, Roland, Dr.-Ing.  
Schellstrasse 1  
D-8000 München 80(DE)

Cache memory consistency control with explicit software instructions.

Memory integrity is maintained in a system with a hierarchical memory using a set of explicit cache control instructions. The caches in the system have two status flags, a valid bit and a dirty bit, with each block of information stored. The operating system executes selected cache control instructions to ensure memory integrity whenever there is a possibility that integrity could be compromised.

21

1	DATA	PHYSICAL TAG	V	D
2				
3				
4				
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
n-1				
n				

**FIG 2**

HEWLETT-PACKARD COMPANY

Pal Alto, USA

EU 012 20

Telefon (089) 4482496

Telefax (089) 4480433

Telex 5214382 pall d

### Cache Memory Consistency Control with Explicit Software Instructions

---

7        Most modern computer systems include a central  
8 processing unit (CPU) and a main memory. The speed at which  
9 the CPU can decode and execute instructions to process data  
10 has for some time exceeded the speed at which instructions  
11 and operands can be transferred from main memory to the CPU.  
12 In an attempt to reduce the problems caused by this  
13 mismatch, many computers include a cache memory or buffer  
14 between the CPU and main memory.

15        Cache memories are small, high-speed buffer memories  
16 used to hold temporarily those portions of the contents of  
17 main memory which are believed to be currently in use by the  
18 CPU. The main purpose of caches is to shorten the time  
19 necessary to perform memory accesses, either for data or  
20 instruction fetch. Information located in cache memory may  
21 be accessed in much less time than that located in main  
22 memory. Thus, a CPU with a cache memory needs to spend far  
23 less time waiting for instructions and operands to be  
24 fetched and/or stored. For such machines the cache memory  
25 produces a very substantial increase in execution speed.

26        A cache is made up of many blocks of one or more words  
27 of data, each of which is associated with an address tag  
28

1 that uniquely identifies which block of main memory it is a  
2 copy of. Each time the processor makes a memory reference,  
3 the cache checks to see if it has a copy of the requested  
4 data. If it does, it supplies the data; otherwise, it gets  
5 the block from main memory, replacing one of the blocks  
6 stored in the cache, then supplies the data to the  
7 processor. See, Smith, A. J., Cache Memories, ACM Computing  
8 Surveys, 14:3 (Sept. 1982), pp. 473-530.

9 Optimizing the design of a cache memory generally has  
10 four aspects:

- 11 (1) Maximizing the probability of finding a memory  
12 reference's target in the cache (the hit ratio),
- 13 (2) minimizing the time to access information that is  
14 indeed in the cache (access time),
- 15 (3) minimizing the delay due to a miss, and
- 16 (4) minimizing the overheads of updating main memory  
17 and maintaining multicache consistency.

18 All of these objectives are to be accomplished under  
19 suitable cost constraints and in view of the inter-  
20 relationship between the parameters.

21 When the CPU executes instructions that modify the  
22 contents of the current address space, those changes must  
23 eventually be reflected in main memory; the cache is only a  
24 temporary buffer. There are two general approaches to  
25 updating main memory: stores can be transmitted directly to  
26 main memory (referred to as write-through or store-through),  
27 or stores can initially modify the data stored in the cache,  
28

1 and can later be reflected in main memory (copy-back or  
2 write-to). The choice between write-through and copy-back  
3 strategies also has implications in the choice of a method  
4 for maintaining consistency among the multiple cache  
5 memories in a tightly coupled multiprocessor system.

6 A major disadvantage to the write-through approach is  
7 that write-through requires a main memory access on every  
8 store. This adds significantly to the relatively slow main  
9 memory traffic load which slows the execution rate of the  
10 processor and which the cache is intended to minimize.  
11 However, when write-through is not used, the problem of  
12 cache consistency arises because main memory does not always  
13 contain an up-to-date copy of all the information in the  
14 system.

15 Input and output between the main memory and peripheral  
16 devices is an additional source of references to the  
17 information in main memory which must be harmonized with the  
18 operation of cache memories. It is important that an output  
19 request stream reference the most current values for the  
20 information transferred. Similarly, it is also important  
21 that input data be immediately reflected in any and all  
22 copies of those lines in memory.

23 There have been several approaches to solving this  
24 problem. One is to direct the I/O stream through the cache  
25 itself. This method is limited to single processor systems.  
26 Further, it interferes significantly with the processor's  
27 use of the cache, both by keeping the cache busy when the

28

1 processor needs it and by displacing blocks of information  
2 currently being used by the processor with the blocks from  
3 the I/O stream. Thus it degrades both the cache access time  
4 and the hit rate. An alternate approach is to use a write-  
5 through policy and broadcast all writes so as to update or  
6 invalidate the target line wherever found. Although this  
7 method accesses main memory instead of the cache, it suffers  
8 from the disadvantages of the write-through strategy,  
9 discussed above. In addition, this hardware intensive  
10 solution is expensive to implement and increases the cache  
11 access cycle time by requiring the cache to check for  
12 invalidation. This is particularly disadvantageous in  
13 multiprocessor systems because every cache memory in the  
14 system can be forced to surrender a cycle to invalidation  
15 lookup whenever any processor in the system performs a  
16 store.

17 Another alternative is to implement a directory to keep  
18 track of the location and status of all copies of each block  
19 of data. The directory can be centralized in main memory or  
20 distributed among the caches, I/O channels and main memory.  
21 This system insures that at any time only one processor or  
22 I/O channel is capable of modifying any block of data. See,  
23 Tang, C.K., Cache Design in the Tightly Coupled  
24 Multiprocessor System, AFIPS Proc., N.C.C., vol. 45, pp.  
25 749-53 (1976). The major disadvantage of the directory  
26 control system is the complexity and expense of the  
27 additional hardware it requires.

1 Finally, if a process r fails, for instance because of  
2 a power interruption, the memory system must assure that the  
3 most current copies of information are stored in main  
4 memory, so that recovery can be more easily accomplished.

5 It is an object of this invention to provide a system  
6 for maintaining the memory integrity and consistency in a  
7 computer system having cache memories, placing the burden of  
8 maintaining integrity on the software, thus allowing the  
9 hardware to remain relatively simple, cheap and fast.

10 It is also an object of this invention to minimize the  
11 impact of the overhead for maintaining memory integrity and  
12 consistency on the operation of the cache memories, so that  
13 the cache access time and miss ratio can be minimized.

14 These and other objects of the invention are  
15 accomplished in a computer having an instruction set  
16 including explicit instructions for controlling the inval-  
17 idation or removal of blocks of data in the cache memories.  
18 Each block of data stored in the caches has two one-bit  
19 status flags, a valid bit to indicate whether the block  
20 contains up-to-date information, and a dirty bit to indicate  
21 whether the data in the block has been stored to by the  
22 processor since it entered the cache. The instruction set  
23 includes instructions for removing a block with a particular  
24 address from the cache and writing it back to memory if  
25 necessary, for removing a block without writeback to main  
26 memory, for suspending execution of instructions until  
27 pending cache control operations are completed, and for

1 efficiently removing and writing back to main memory all  
2 "dirty" blocks in the cache in case of a processor failure.  
3 The operating system software invokes these instructions in  
4 situations which could result in inconsistent or stale data  
5 in the cache memories.

6  
7  
8  
9 Figure 1 is a schematic block diagram of a computer  
10 system which incorporates the invention.

11 Figure 2 is a schematic illustration of a cache memory  
12 constructed in accordance with the invention.

13 Figure 3 is a schematic illustration of an alternative  
14 form of cache memory constructed in accordance with the  
15 invention. ....

16  
17  
18  
19  
20 A computer system which operates according to the  
21 invention is schematically illustrated in Figure 1. The  
22 main processor 11, often referred to as the CPU, .  
23 communicates with main memory 13 and input/output channel 15  
24 via memory bus 17. The main processor includes a processor  
25 19 which fetches, decodes and executes instructions to  
26 process data. Data and instructions are stored in main  
27 memory 13, transferred to processor 19 when they are  
28

1 r requested during th xecution of a program or routine and  
2 returned to main memory 13 after the program or routine has  
3 been completed.

4 Access to main memory 13 is relatively slow compared  
5 with the operation of processor 19. If processor 19 had to  
6 wait for main memory access to be completed each time an  
7 instruction or data was needed, its execution rate would be  
8 reduced significantly. In order to provide access times  
9 which more closely match the needs of the processor, cache  
10 21, which may be referred to as a buffer memory, stores a  
11 limited number of instructions and data. Since cache 21 is  
12 much smaller than main memory 13 it can be economically  
13 built to have higher access rates.

14 The operating system software for the computer, rather  
15 than the hardware of the component units, is responsible for  
16 maintaining the integrity and consistency of the memory. In  
17 order to accomplish this, the operating system invokes  
18 explicit control instructions included in the computer's  
19 instruction set.

20 To explain the system of the invention more completely,  
21 an understanding of the structure of cache memory 21 is  
22 necessary. The entries of the array in cache memory 21 are  
23 illustrated in Figure 2. Cache 21 comprises an array of  
24 locations labeled with an index 31 which store data 33 and a  
25 physical page tag 35 which corresponds to the physical page  
26 number of the location of the copy of the data in main  
27 memory.

1 In addition to the data 33 and tags 35 stored in the  
2 cache, each block has associated with it two one-bit status  
3 flags, "valid" and "dirty". The valid bit 37 is set if and  
4 only if that block has valid data, i.e., up-to-date data.

5 The dirty bit 39 is set if the processor has stored to  
6 the address since it has been brought into the cache.  
7 Unless cache 21 updates main memory 13 every time processor  
8 19 does a store (write-through), the cache has more up-to-  
9 date data for a block than main memory has. Dirty bit 39  
10 serves to indicate that main memory 13 must be updated by  
11 writing the data in the block in cache 21 back to main  
12 memory 13 when the block is swapped out of the cache.

13 Cache 21 can also be divided into two sections, one for  
14 data and another for instructions, as illustrated in Figure  
15 3. For many computer architectures, this split cache  
16 provides performance advantages. Both the instruction cache  
17 41 and the data cache 51 have structures similar to that of  
18 the unified cache described above. Instruction cache 41 has  
19 an array of locations labeled with an index 43. Each  
20 location stores an entry comprising: a physical page tag  
21 45, an instruction 46 and a valid bit 47. Data cache 51  
22 has an array of locations labelled with an index 53. Each  
23 location stores an entry comprising: a physical tag 55, a  
24 block of data 56, a valid bit 57 and a dirty bit 58.  
25 Although this cache organization provides certain  
26 advantages, it also requires additional control instruc-  
27 tions. In particular, instructions may be modified and

28

1 copies may then appear in both sections of the cache. The  
2 operating system must therefore flush blocks from the  
3 instruction cache 41 and from data cache 51 back to main  
4 memory 13 to insure consistency.

5 The operating system performs the required memory  
6 maintenance functions using six instructions: Flush Data  
7 Cache, Purge Data Cache, Flush Instruction Cache, Flush Data  
8 Cache Entry, Flush Instruction Cache Entry and Synchronize  
9 Caches.

10 The Flush Data Cache (FDC) instruction sets the  
11 addressed data cache valid bit to "invalid" if the data  
12 address hits the data cache. The block of data at the given  
13 address is removed from the cache and written back to the  
14 main memory if the dirty bit is set.

15 The Purge Data Cache (PDC) instruction sets the  
16 addressed data cache valid bit to "invalid" if the data  
17 address hits the cache. The block of data at the given  
18 address is removed from the cache and no write-back is  
19 performed.

20 The Flush Instruction Cache (FIC) instruction sets the  
21 addressed instruction cache valid bit to "invalid" if the  
22 address hits the cache. The instruction at the given  
23 address is removed from the cache.

24 The Flush Data Cache Entry (FDCE) instruction is a  
25 special kind of flush that can be used in a routine to flush  
26 the entire cache, for example in the event of a processor  
27 failure. This routine is implementation dependent. For a

28

1 multiple-set cache, the routine steps through the index  
2 range of cache once for each set. The FDCE instruction  
3 flushes a block of data and sets the addressed data cache  
4 valid bit to "invalid" whether or not there is a hit at the  
5 cache index. That is, the block is written back to main  
6 memory if and only if it is valid and dirty, without  
7 comparing the cache tag to any requested address.

8 The Flush Instruction Cache Entry (FICE) instruction  
9 accomplishes the same function in the instruction cache as  
10 the FDCE instruction accomplishes in the data cache.

11 The Synchronize Caches (SYNC) instruction suspends  
12 instruction execution by the processor until the completion  
13 of all instruction cache and data cache operations. This  
14 guarantees that any reference to data will await the  
15 completion of the cache operations required to ensure the  
16 integrity of that data.

17 The operation of the system is illustrated by the  
18 following examples. The operating system controls access to  
19 main memory by the processor and by the peripheral devices  
20 attached to I/O channel 15.

21 When data is to be read into main memory 13 from an  
22 external device through I/O channel 15, the operating system  
23 must insure that the addresses into which or from which the  
24 data is transferred do not overlap areas mapped into either  
25 data or instruction caches. In order to clear any stale  
26 data out of the caches, before the I/O is performed, the  
27 system broadcasts to each cache the FDC and FIC instruction

28

1 ver the range of addresses into which the input data is to  
2 be mapped.

3 When data is to be read out of main memory to an  
4 external device through I/O channel 15, the operating system  
5 must insure that the addresses from which the data is  
6 transferred do not overlap areas mapped into data caches, so  
7 that the most up-to-date data is transferred. In order to  
8 update main memory with the data in the caches that has been  
9 modified by the processors, the system broadcasts to each  
10 cache the FDC instruction for the range of addresses from  
11 which the output data is to be read. The FDC instruction  
12 causes the cache to write any dirty blocks back to main  
13 memory.

14 In a virtual memory system, whenever a page or segment  
15 is moved from main memory 13 to a peripheral memory (eg., a  
16 disc memory) connected to I/O channel 15, the data from the  
17 page or segment must be flushed from all caches. The  
18 operating system broadcasts to the caches the FDC and FIC  
19 instruction over the range of addresses included in the page  
20 or segment. When a page or segment is destroyed, for  
21 example because of program termination, the data must be  
22 removed from the cache but need not be stored. In this  
23 instance, the operating system uses the PDC and FIC  
24 instructions. No flush or purge operations are needed when  
25 a page or segment is created or brought in from a peripheral  
26 memory because the addresses into which it is mapped will  
27 have just been flushed or purged during the removal of the  
28

1 previous page or segment to make room for the new page or  
2 segment.

3       In order to accommodate programs with self-modifying  
4 code, the operating system must remove from the caches any  
5 stale copies of the modified instruction to guarantee that  
6 only the new version of the instruction will be executed.  
7 After the modification of the instruction has been done in  
8 data cache 51, the operating system uses the FDC instruction  
9 to force the modified copy out to main memory, uses the FIC  
10 instruction to remove any stale copy of the instruction from  
11 instruction cache 41, then executes the SYNC instruction to  
12 insure that the modified instruction is not invoked until  
13 the FDC and FIC instructions have been completed.

14       In the event of a processor failure, for example caused  
15 by a power failure, the modified blocks of data residing in  
16 the caches must be written back to main memory. The  
17 operating system can accomplish this in a minimal amount of  
18 time with the FDCE and FICE instructions. A routine using  
19 the FDCE and FICE instructions flushes the caches quickly  
20 because by stepping through the index range of the caches  
21 rather than using the address space which is much larger.  
22 As the routine steps through the caches, only the blocks  
23 that are valid and dirty are written back to main memory 13.

24  
25  
26  
27  
28

1. A computer system having a multi-level memory hierarchy and means for maintaining the integrity of the blocks of information stored at different levels in the hierarchy, characterized by
- 5 by
- a processor (19) for executing instructions and processing data;
- memory (13) for storing instructions and data;
- an I/O channel (15) connected to the memory
- 10 (13) for transferring data and instructions into and out of the memory (13);
- a cache (21) connected between the processor (19) and the memory (13) for storing selected blocks of information from the memory (13) for
- 15 use by the processor (19), and having associated with each stored block a valid status flag and a dirty status flag;
- a set of instructions for providing explicit control of the removal of blocks of data from
- 20 the cache (21); and
- an operating system capable of causing the execution of certain of the instructions from the instruction set to ensure the consistency of the information stored in the cache (21) with the information
- 25 transferred into and out of memory (13).
2. Computer system according to claim 1, characterized in that the instruction set comprises Flush Data Cache, Purge Data Cache, Flush
- 30 Instruction Cache, Flush Data Cache Entry, Flush Instruction Cache Entry and Synchronize Caches instructions; and in that prior to transfer of data or instructions into or out of memory (13) via the I/O channel (15), the operating
- 35 system broadcasts the Flush Data Cache and Flush Instruction Cache instructions to the cache

(21) over the range of addresses into which or out of which data is transferred.

3. Computer system according to claim 2, characterized in that virtual memory (13) is used; in that, when a page or a segment is removed from memory (13), the operating system broadcasts the Flush Data Cache and Flush Instruction Cache instructions to the cache (21) over the range of addresses in the page or segment; and in that, when a page or segment is destroyed, the operating system broadcasts the Purge Data Cache and Flush Instruction Cache instructions to the cache (21) over the range of addresses in the page or segment.
4. Computer system according to claim 2 or 3, characterized in that the cache (21) is divided into two segments (41,51), a data cache (51) for storing data and an instruction cache (41) for storing instructions; that instructions can be modified when stored as part of a block of data in the data cache (51); and in that after modification of an instruction, the operating system issues the Flush Data Cache instruction to the data cache (51) for the address of the block including the modified instruction, issues the Flush Instruction Cache instruction to the instruction cache (41) for the address of the modified instruction and then executes the Synchronize Caches instruction.
5. Computer system according to any of claims 2 to 4, characterized in that in the event of a processor failure, the operating system executes a routine including the Flush Data Cache Entry and Flush Instruction Cache

Entry instructions over the index range for the data cach (51) and for the instruction cache (41).

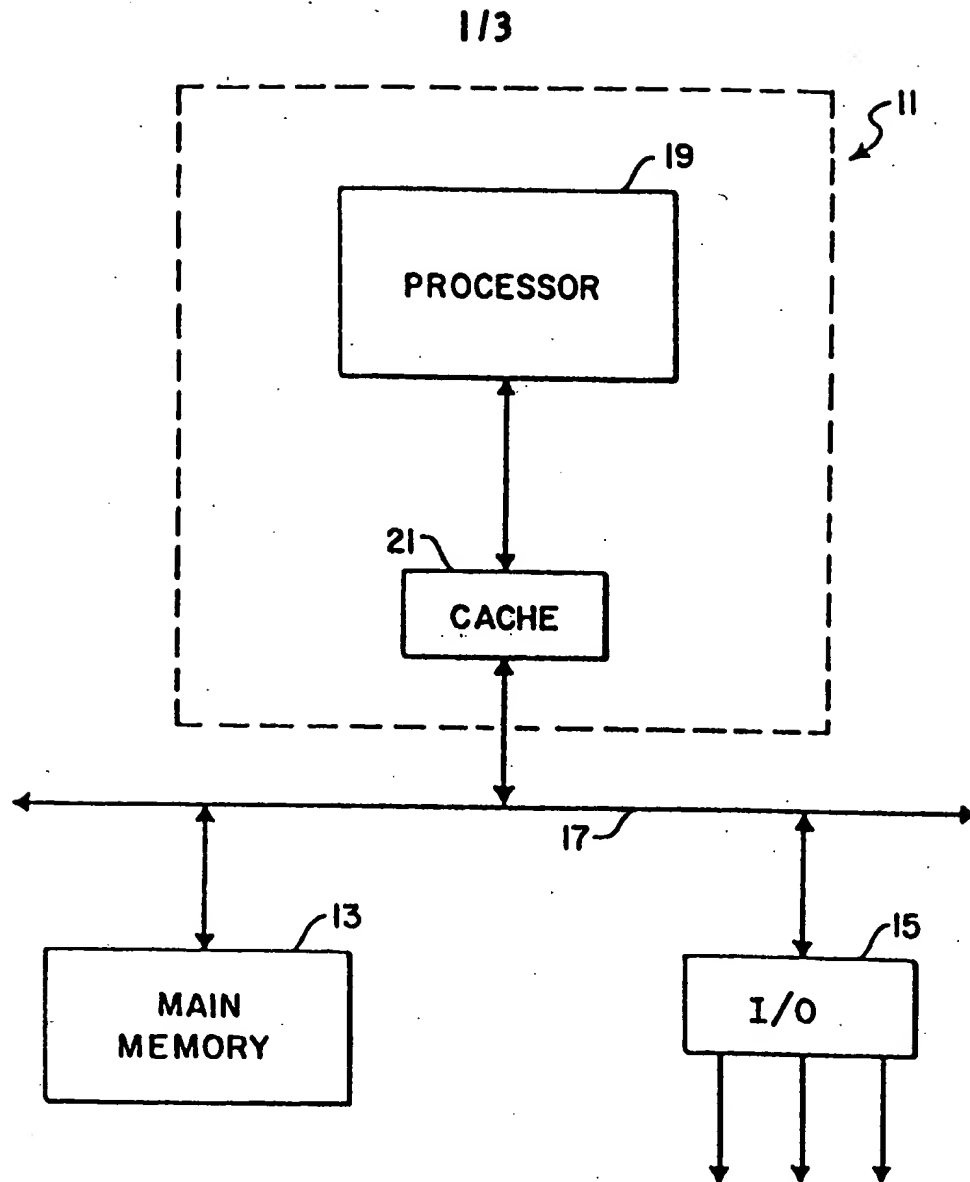


FIG 1

1135

0210384

2/3

21 ↘

1	DATA	PHYSICAL TAG	V	D
2				
3				
4				
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
n-1				
n				

31      33      35      37      39

FIG 2

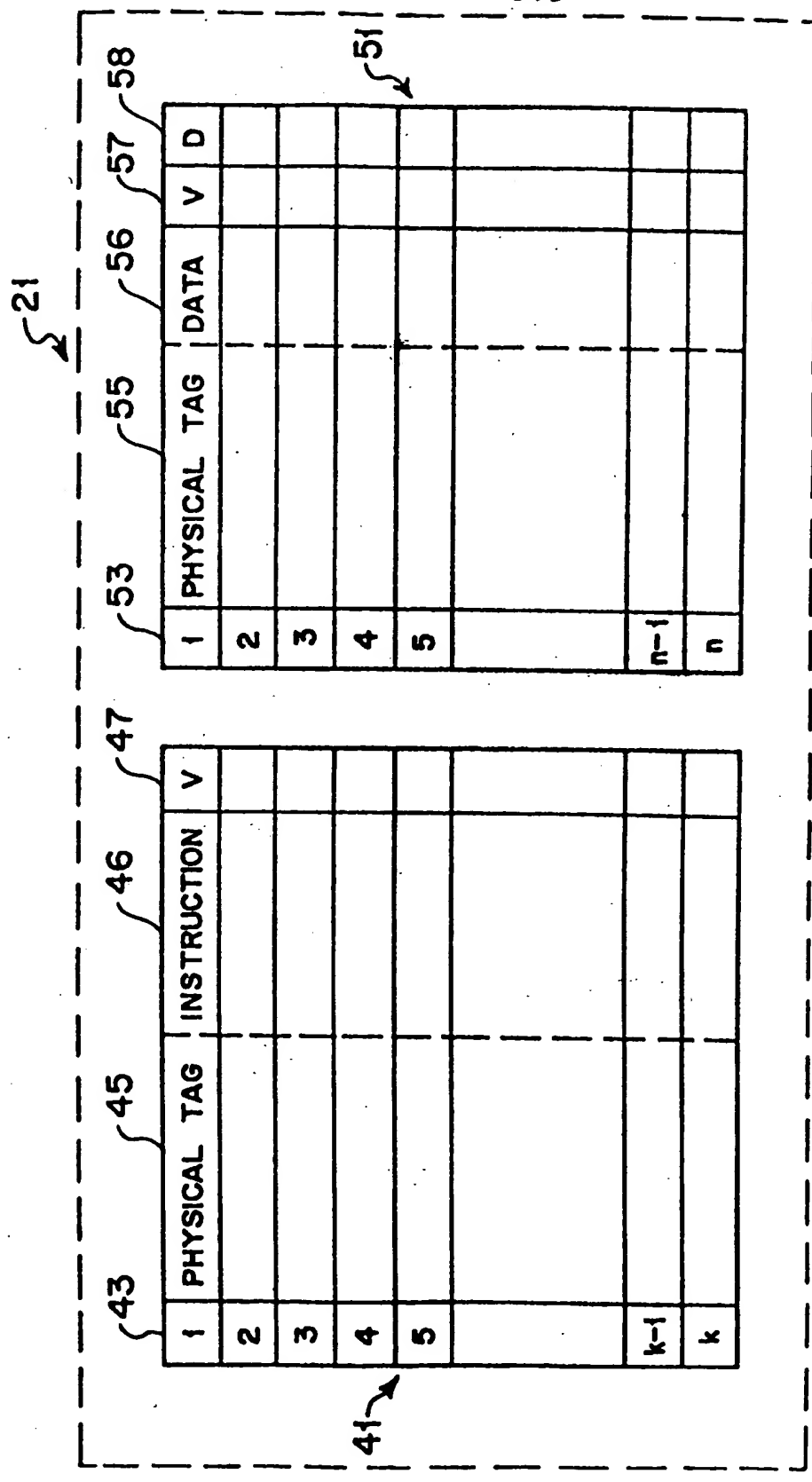


FIG 3



European Patent  
Office

# EUROPEAN SEARCH REPORT

0210384

Application number

EP 86 10 7713

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl. 4)
Y	US-A-3 771 137 (BARNER et al.) * Figures 1-3; column 3, line 49 - column 5, line 52 *	1	G 06 F 12/08
A	---	4	
Y	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 24, no. 7A, December 1981, pages 3128, 3129, New York, US; J.F. COURT et al.: "Technique for improved channel performance" * Whole document *	1	
A	IDEM	2	
A	US-A-3 845 474 (LANGE et al.) * Figure 8; column 11, line 38 - column 12, line 41 *	2, 3, 5	TECHNICAL FIELDS SEARCHED (Int. Cl. 4) G 06 F 12/08
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 23, no. 7B, December 1980, page 3329, New York, US; B.B. MOORE et al.: "Vary storage physical on/off-line in a non-store-through cache system" * Whole document *	1, 2	
---		-/-	
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 02-10-1986	Examiner LEDROUT P.
<p><b>CATEGORY OF CITED DOCUMENTS</b></p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application I : document cited for other reasons &amp; : member of the same patent family, corresponding document</p>			



European Patent  
Office

# EUROPEAN SEARCH REPORT

0210384

Application number

EP 86 10 7713

DOCUMENTS CONSIDERED TO BE RELEVANT			Page 2
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl. 4)
A	CONFERENCE PROCEEDINGS OF THE 11TH ANNUAL SYMPOSIUM ON COMPUTER ARCHITECTURE, Ann Arbor, Michigan, US, 5th-7th June 1984, pages 348-354, IEEE, New York, US; M.S. PAPAMARCOS et al.: "A low-overhead coherence solution for multiprocessors with private cache memory" * Pages 348-350 *	1	
A	--- EP-A-O 145 594 (FUJITSU) * Figure 3; page 5, line 12 - page 7, line 17 *	2,4	
A	--- EP-A-O 052 370 (HITACHI) * Figure 1; page 13, line 12 - page 15, line 11 *	2,4	
			TECHNICAL FIELDS SEARCHED (Int. Cl. 4)
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 02-10-1986	Examiner LEDROUT P.
<p><b>CATEGORY OF CITED DOCUMENTS</b></p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technical background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons &amp; : member of the same patent family, corresponding document</p>			